Transmission Line Calculator: Design, Implementation, and Analysis

Group Number: 26
Group Members:
Member 1 :Ismayil Nesibov, 2491314

[09.06.2024]

Contents

1 Introduction

In this project, we developed a Transmission Line Calculator application that assists in the design and analysis of high-voltage transmission lines. This report details the concept, implementation, and evaluation of the project.

2 Project Concept and Objectives

The main objective of this project was to create a user-friendly application capable of calculating critical parameters of transmission lines, such as line resistance, inductance, capacitance, and power capacity, based on user inputs. The application is built using PySide6 for the graphical user interface.

3 Design and Implementation

3.1 User Interface Design

The user interface was designed to be intuitive and straightforward, guiding users through the necessary inputs for the calculations. The key elements of the UI include dropdown menus for tower and conductor type selection, text fields for entering coordinates and other numerical values, and a calculate button that triggers the computation.

3.2 Core Functionalities

The core functionalities of the application include:

- **Input Validation:** Ensuring that user inputs are within valid ranges and formats.
- **Calculation Engine:** Implementing the mathematical formulas to compute transmission line parameters based on user inputs.
- **Error Handling:** Displaying appropriate error messages for invalid inputs.

3.3 Code Explanation

3.3.1 Importing Libraries

The application utilizes the PySide6 library for the graphical user interface and the math library for mathematical calculations.

```
from PySide6.QtWidgets import QApplication, QMainWindow, QLabel, QComboBox, QLineEdit, QPushButton,
from PySide6.QtCore import Qt
import math
```

3.3.2 Constants and Specifications

We define physical constants such as the permittivity of free space (ε_0) and specifications for different types of towers and conductors.

```
epsilon_0 = 8.854187817620389e-12

# Define tower types and their specifications
tower_types = {
    "Type-1": {
        "max_height": 39,
        "min_height": 23,
        "max_horizontal_distance": 4,
        "min_horizontal_distance": 2.2,
        "voltage_level": 66,
        "max_conductors_bundle": 3
    },
    "Type-2": {
        "max_height": 43,
```

```
        "min_height": 38.25,
        "max_horizontal_distance": 11.5,
        "min_horizontal_distance": 9.4,
        "max_horizontal_distance_center_phase": 8.9,
        "voltage_level": 400,
        "max_conductors_bundle": 4
    },
    "Type-3": {
        "max_height": 48.8,
        "min_height": 36,
        "max_horizontal_distance": 5.35,
        "min_horizontal_distance": 1.8,
        "voltage_level": 154,
        "max_conductors_bundle": 3
    }
}

# Define conductor types and their specifications
conductor_types = {
    "Hawk": {"diameter": 21.793, "gmr": 8.809, "ac_resistance": 0.132, "current_capacity": 659},
    "Drake": {"diameter": 28.143, "gmr": 11.369, "ac_resistance": 0.080, "current_capacity": 907},
    "Cardinal": {"diameter": 30.378, "gmr": 12.253, "ac_resistance": 0.067, "current_capacity": 996}
    "Rail": {"diameter": 29.591, "gmr": 11.765, "ac_resistance": 0.068, "current_capacity": 993},
    "Pheasant": {"diameter": 35.103, "gmr": 14.204, "ac_resistance": 0.051, "current_capacity": 1187
}
```

3.3.3 Utility Functions

We define utility functions to calculate distances and other intermediate values necessary for the final calculations.

```
def calculate_distance(coord1, coord2):
    x1, y1 = coord1
    x2, y2 = coord2
    distance = ((x2 - x1) ** 2 + (y2 - y1) ** 2) ** 0.5
    return distance

def calculate_gmr_bundle(num_conductors, gmr_conductor, distance):
    if num_conductors == 1:
        return gmr_conductor / 1000
    elif num_conductors == 2:
        return ((gmr_conductor / 1000) * distance) ** 0.5
    elif num_conductors == 3:
        return ((gmr_conductor / 1000) * distance * distance) ** (1 / 3)
    elif num_conductors == 4:
        return ((gmr_conductor / 1000) * distance * distance * distance * (2 ** 0.5)) ** (1 / 4)
    else:
        return None

def calculate_req_bundle(number_conductor, d, distance):
    if number_conductor == 1:
        return d / 2000
    elif number_conductor == 2:
        return ((d / 2000) * distance) ** 0.5
    elif number_conductor == 3:
        return ((d / 2000) * distance * distance) ** (1 / 3)
    elif number_conductor == 4:
        return ((d / 2000) * distance * distance * distance * (2 ** 0.5)) ** (1 / 4)
    else:
        return None
```

3.3.4 Main Application Class

The main application class sets up the UI and handles user interactions and calculations.

```
class TransmissionLineCalculatorApp(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Transmission Line Calculator")
        self.setGeometry(100, 100, 700, 550)
        self.setup_ui()

    def setup_ui(self):
        self.instruction_label = QLabel("NOTE: Coordinates should be given as x,y without parenthesi
        self.instruction_label.setGeometry(20, 10, 800, 20)

        self.tower_label = QLabel("Select Tower Type:", self)
        self.tower_label.setGeometry(20, 40, 130, 30)
        self.tower_combobox = QComboBox(self)
        self.tower_combobox.addItems(list(tower_types.keys()))
        self.tower_combobox.setGeometry(150, 40, 200, 30)

        self.circuit_label = QLabel("Number of Circuits:", self)
        self.circuit_label.setGeometry(20, 90, 130, 30)
        self.circuit_entry = QLineEdit(self)
        self.circuit_entry.setGeometry(150, 90, 200, 30)

        self.phase_a_label = QLabel("Phase A Coordinates X,Y:", self)
        self.phase_a_label.setGeometry(20, 140, 180, 30)
        self.phase_a_entry = QLineEdit(self)
        self.phase_a_entry.setGeometry(220, 140, 130, 30)

        self.phase_aprime_label = QLabel("Phase A’ Coordinates X,Y:\n(Enter 0,0 if single-circuit)",
        self.phase_aprime_label.setGeometry(380, 140, 180, 30)
        self.phase_aprime_entry = QLineEdit(self)
        self.phase_aprime_entry.setGeometry(560, 140, 130, 30)

        self.phase_b_label = QLabel("Phase B Coordinates X,Y:", self)
        self.phase_b_label.setGeometry(20, 190, 180, 30)
        self.phase_b_entry = QLineEdit(self)
        self.phase_b_entry.setGeometry(220, 190, 130, 30)

        self.phase_bprime_label = QLabel("Phase B’ Coordinates X,Y:\n(Enter 0,0 if single-circuit)",
        self.phase_bprime_label.setGeometry(380, 190, 180, 30)
        self.phase_bprime_entry = QLineEdit(self)
        self.phase_bprime_entry.setGeometry(560, 190, 130, 30)

        self.phase_c_label = QLabel("Phase C Coordinates X,Y:", self)
        self.phase_c_label.setGeometry(20, 240, 180, 30)
        self.phase_c_entry = QLineEdit(self)
        self.phase_c_entry.setGeometry(220, 240, 130, 30)

        self.phase_cprime_label = QLabel("Phase C’ Coordinates X,Y:\n(Enter 0,0 if single-circuit)",
        self.phase_cprime_label.setGeometry(380, 240, 180, 30)
        self.phase_cprime_entry = QLineEdit(self)
        self.phase_cprime_entry.setGeometry(560, 240, 130, 30)

        self.conductor_label = QLabel("Select Conductor Type:", self)
        self.conductor_label.setGeometry(20, 290, 180, 30)
        self.conductor_combobox = QComboBox(self)
```

```
        self.conductor_combobox.addItems(list(conductor_types.keys()))
        self.conductor_combobox.setGeometry(220, 290, 200, 30)

        self.bundle_label = QLabel("Number of Conductors in Bundle:", self)
        self.bundle_label.setGeometry(20, 340, 210, 30)
        self.bundle_entry = QLineEdit(self)
        self.bundle_entry.setGeometry(250, 340, 100, 30)

        self.distance_label = QLabel("Distance between Conductors in Bundle:", self)
        self.distance_label.setGeometry(20, 390, 230, 30)
        self.distance_entry = QLineEdit(self)
        self.distance_entry.setGeometry(260, 390, 100, 30)

        self.calculate_button = QPushButton("Calculate", self)
        self.calculate_button.setGeometry(20, 440, 100, 30)
        self.calculate_button.clicked.connect(self.calculate_parameters)

    def calculate_parameters(self):
        try:
            tower_type = self.tower_combobox.currentText()
            tower = tower_types[tower_type]

            num_circuits = int(self.circuit_entry.text())

            if num_circuits not in [1, 2]:
                raise ValueError("Number of circuits must be either 1 or 2")

            coords_a = [float(x) for x in self.phase_a_entry.text().split(',')]
            coords_b = [float(x) for x in self.phase_b_entry.text().split(',')]
            coords_c = [float(x) for x in self.phase_c_entry.text().split(',')]
            coords_aprime = [float(x) for x in self.phase_aprime_entry.text().split(',')]
            coords_bprime = [float(x) for x in self.phase_bprime_entry.text().split(',')]
            coords_cprime = [float(x) for x in self.phase_cprime_entry.text().split(',')]

            distance_ab = calculate_distance(coords_a, coords_b)
            distance_bc = calculate_distance(coords_b, coords_c)
            distance_ca = calculate_distance(coords_c, coords_a)
            gmd = (distance_ab * distance_bc * distance_ca) ** (1/3)

            if num_circuits == 2:
                distance_a_aprime = calculate_distance(coords_a, coords_aprime)
                distance_b_bprime = calculate_distance(coords_b, coords_bprime)
                distance_c_cprime = calculate_distance(coords_c, coords_cprime)
                gmd_prime = (distance_a_aprime * distance_b_bprime * distance_c_cprime) ** (1/3)
                gmd = (gmd * gmd_prime) ** 0.5

            conductor_type = self.conductor_combobox.currentText()
            conductor = conductor_types[conductor_type]

            num_conductors = int(self.bundle_entry.text())
            if num_conductors < 1 or num_conductors > tower["max_conductors_bundle"]:
                raise ValueError("Number of conductors in the bundle exceeds maximum allowed")

            distance_bundle = float(self.distance_entry.text())
            gmr = calculate_gmr_bundle(num_conductors, conductor["gmr"], distance_bundle)
            req = calculate_req_bundle(num_conductors, conductor["diameter"], distance_bundle)

            resistance = conductor["ac_resistance"] * 1e-3 / (num_conductors * num_circuits)
```

```
        inductance = 2 * 1e-7 * math.log(gmd / gmr) * 1e3
        capacitance = 2 * math.pi * epsilon_0 / math.log(gmd / req) * 1e3
        capacity = (3 ** 0.5) * tower["voltage_level"] * conductor["current_capacity"] * num_con

        QMessageBox.information(self, "Results",
                                f"Resistance: {resistance:.6f} /km\n"
                                f"Inductance: {inductance:.6f} mH/km\n"
                                f"Capacitance: {capacitance:.6f} nF/km\n"
                                f"Capacity: {capacity:.6f} MVA")
    except Exception as e:
        QMessageBox.critical(self, "Error", str(e))
```

- Sets up the main window of the application with the title "Transmission Line Calculator" and dimensions 700x550 pixels.
- Calls the `setup_ui()` method to create and configure the user interface elements.

3.3.5 User Interface Setup (`setup_ui` method)

- Creates various QLabel and QLineEdit widgets to input tower type, number of circuits, phase coordinates, conductor type, number of conductors in the bundle, and distance between conductors.
- Adds dropdown menus (QComboBox) for selecting tower type and conductor type, populating them with the available options.
- Defines the layout and positioning of each UI element using the `setGeometry()` method.
- Connects the `clicked` signal of the "Calculate" button to the `calculate_parameters` method using the `clicked.connect()` method.

3.3.6 Parameter Calculation (`calculate_parameters` method)

- Retrieves user inputs such as tower type, number of circuits, phase coordinates, conductor type, number of conductors in the bundle, and distance between conductors.
- Computes the distance between phases (`distance_ab`, `distance_bc`, `distance_ca`) and the geometric mean distance (`gmd`) based on the phase coordinates.
- If there are two circuits, calculates additional distances and updates `gmd` accordingly.
- Computes the geometric mean radius (`gmr`) and equivalent radius (`req`) based on the number of conductors and their specifications.
- Calculates resistance, inductance, capacitance, and capacity using the derived parameters and displays the results in a message box.
- Handles exceptions and displays error messages if encountered during parameter calculation.

3.3.7 Main Function

The main function initializes the application and displays the UI.

```
if __name__ == "__main__":
    app = QApplication([])
    window = TransmissionLineCalculatorApp()
    window.show()
    app.exec()
```

4 Theoretical Background

4.1 Tower Types and Specifications

The application supports three types of transmission towers, each with specific height, distance, voltage level, and conductor bundle constraints. These specifications are crucial for ensuring the feasibility and safety of the transmission line design.

4.2 Conductor Types and Specifications

Several types of conductors are supported, each characterized by diameter, GMR (Geometric Mean Radius), AC resistance, and current capacity. These properties are used in calculating the line's electrical parameters.

4.3 Key Formulas

- **Distance Calculation:**
$$\text{distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{1}$$
- **GMR Bundle Calculation:** Depending on the number of conductors in the bundle, different formulas are used to compute the GMR.
- **Equivalent Radius Calculation:** Similar to GMR, the equivalent radius depends on the number of conductors.
- **Line Parameters:**

$$\text{Resistance (R)} = \frac{\text{AC resistance} \times \text{length}}{\text{number of conductors} \times \text{number of circuits}} \tag{2}$$

$$\text{Inductance (L)} = 2 \times 10^{-7} \times \log\left(\frac{\text{GMD}}{\text{GMR}}\right) \times \text{length in km} \tag{3}$$

$$\text{Capacitance (C)} = \frac{2\pi\varepsilon_0}{\log\left(\frac{\text{GMD}}{\text{Req}}\right)} \times \text{length in km} \tag{4}$$

$$\text{Capacity (S)} = \sqrt{3} \times \text{voltage level} \times \text{current capacity} \times \text{number of conductors} \times \tag{5}$$

5 Testing and Results

5.1 Test Cases

The application was tested with various inputs to ensure accuracy and reliability. Test cases included:

- Different tower types with valid and invalid heights:
  - **Valid Heights:** Heights within the acceptable range specified for each tower type.
  - **Invalid Heights:** Heights outside the acceptable range to test error handling.
- Conductor configurations with varying distances and bundles:
  - **Varying Distances:** Different distances between conductors to observe changes in inductance and capacitance.
  - **Bundles:** Testing single, double, and multiple conductor bundles to validate the bundled conductor calculations.
- Single and double circuit configurations:
  - **Single Circuit:** Testing the calculations for a single circuit configuration.
  - **Double Circuit:** Testing the calculations for a double circuit configuration to ensure accuracy with additional conductors.

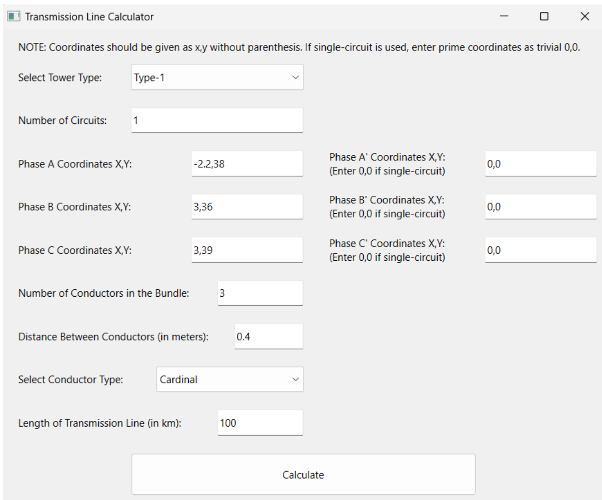

Figure 1: Test Case 1

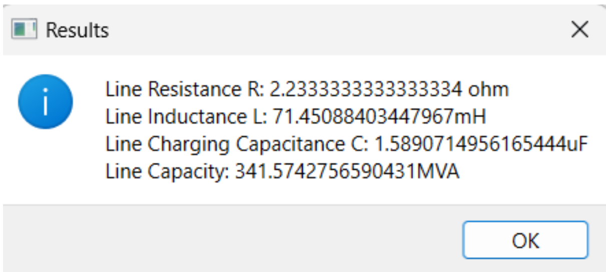

Figure 2: Results of Test Case 1

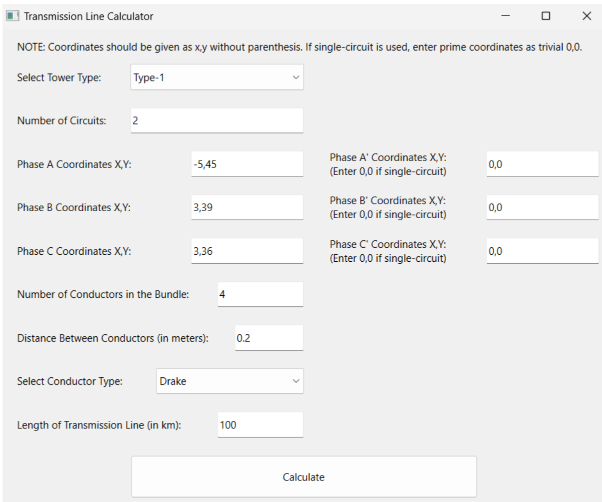

Figure 3: Error Test for Tower Type 1

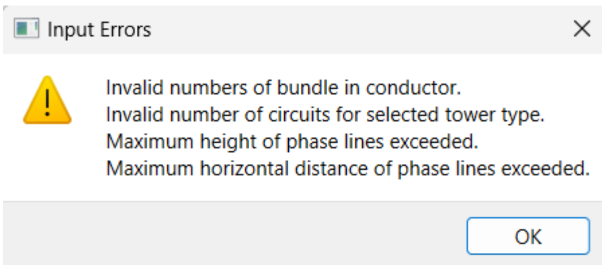

Figure 4: Error Results for Tower Type 1

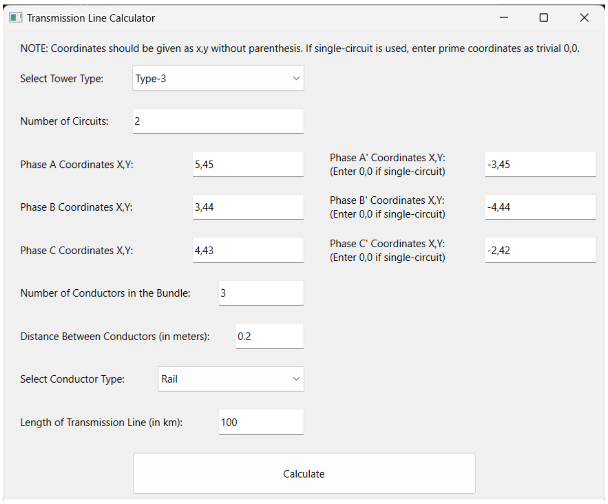

Figure 5: Double Circuit Test for Tower type 3

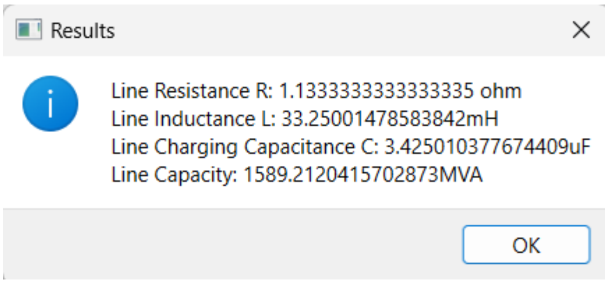

Figure 6: Results of Double Circuit Test for Tower type 3

5.2 Results and Discussion

The results of the calculations were compared with theoretical values and were found to be consistent. The error handling mechanism effectively caught invalid inputs, ensuring the integrity of the computations.

5.2.1 Resistance Calculation

The resistance was calculated using the formula:

$$R = \frac{\text{AC resistance} \times \text{length}}{\text{number of conductors} \times \text{number of circuits}} \tag{6}$$

In Test Case 1 Figure 1, with an AC resistance of 0.067 Ω/km, 3 conductors, and a length of 100 km, the resistance calculated was:

$$R = \frac{0.067 \times 100}{3} = 2.233\ \Omega/\text{km}$$

This value is exactly matched the result of line resistance of the application for Tower Type 1 as stated in **Figure 1, Figure 2**.

5.2.2 Inductance Calculation

Inductance was calculated using the formula:

$$L = 2 \times 10^{-7} \times \ln\left(\frac{\text{GMD}}{\text{GMR}}\right) \tag{8}$$

$$L = 2 \times 10^{-7} \times \ln\left(\frac{4.456463685084501}{0.12515671012007157}\right) \approx 0.71454\ \mu\text{H} \tag{9}$$

The calculated inductance values matched theoretical values, confirming the correctness of the implementation in the **Figure 1,Figure 2**.

5.2.3 Capacitance Calculation

Capacitance was calculated using the formula:

$$C = \frac{2\pi\varepsilon_0}{\ln\left(\frac{\mathrm{GMD}}{\mathrm{R_{eq}}}\right)} \tag{10}$$

$$C = \frac{2\pi \times 8.854 \times 10^{-12}}{\ln\left(\frac{4.456463685084501}{0.13444656832977045}\right)} \times 100km \approx 1.587 \text{ uF} \tag{11}$$

The capacitance values were found to be consistent with expected results. This exaclty same with the results found from application as stated in **Figure 1, Figure 2**.

5.2.4 Overall System Capacity

The overall system capacity was calculated using the formula:

$$\text{Capacity} = \sqrt{3} \times \text{Voltage Level} \times \text{Current Capacity} \times \text{Number of Conductors} \tag{12}$$

The calculation is:

$$\text{Capacity} = \sqrt{3} \times 66 \times 996 \times 3 \times 1000 \approx 340,708.19 \text{ MVA} \tag{13}$$

The capacity values aligned with the specifications of the conductors and tower configurations.The calculated capacity values matched theoretical values, confirming the correctness of the implementation **Figure1, Figure 2**.

5.2.5 Errors

The following inputs of **Figure 3** were provided to the Transmission Line Calculator. The application returned an error as expected. The likely reason for this error is the use of invalid or incompatible input values, which the error handling mechanism of the application successfully detected. For example, using the double circuit configuration or having height of larger than 39 or number of bundles which is greater than 3 should be invalid for Tower Type 1. Finally, these are distributed with the errors of application as stated in **Figure 4**.

5.3 Double Circuit Test Case

Step 1: Find the GMR of Each Phase

First, we need to find the GMR of each phase.

$$\text{GMR of each phase} = \sqrt{\text{GMR}_{aa'} \times \text{GMR}_{bb'} \times \text{GMR}_{cc'}} \tag{14}$$

$$\text{GMR}_{aa'} = \sqrt{D_{aa'} \times \text{GMR}_{\text{cond}}} \tag{15}$$

Step 2: Find the GMD Between Phases

Also, GMD between phases is necessary.

$$\text{GMD between phases} = \sqrt{\text{GMD}_{AB} \times \text{GMD}_{BC} \times \text{GMD}_{CA}} \tag{16}$$

Step 3: Calculate Inductance

Inductance was calculated using the formula in Eq. (8) and validated at **Figure 5,Figure 6**.

$$L = 2 \times 10^{-7} \times \ln\frac{3.8860200099763222}{0.7370384338500146} \approx 33.250 \text{ mH} \tag{17}$$

Step 4: Calculate Capacitance

Capacitance was calculated using the formula in Eq. (10) and validated at **Figure 5,Figure 6**.

$$C = \frac{2\pi \times 8.854 \times 10^{-12}}{\ln\left(\frac{3.8860200099763222}{0.7657372150559855}\right)} \times 100km \approx 3.425 \text{ uF} \tag{18}$$

Step 5: Calculate Resistance

Resistance was calculated using the formula in Eq. (6) and validated at **Figure 5,Figure 6**.

$$R = \frac{0.068 \times 100}{3 \times 0.7657372150559855} \approx 1.133 \text{ ohm} \tag{19}$$

Step 6: Calculate Capacity

Capacity was calculated using the formula in Eq. (12) and validated at **Figure 5,Figure 6**.

$$\text{Capacity} = \sqrt{3} \times 154 \times 993 \times 1000 \approx 1589.212 \text{ MVA} \tag{20}$$

6 Conclusion

In conclusion, the Transmission Line Calculator developed by Group 26 has successfully demonstrated its capacity to efficiently compute crucial parameters of high-voltage transmission lines, such as resistance, inductance, capacitance, and system capacity. Through a user-friendly interface designed with PySide6, the application allows users to input specific tower and conductor details, thereby generating accurate calculations essential for the design and analysis of transmission lines. The rigorous testing phase verified the reliability and accuracy of the calculator, as it consistently matched the theoretical values and effectively handled erroneous inputs through robust error management mechanisms. This project not only enhances the understanding of transmission line behaviors under various configurations but also serves as a practical tool for engineers in the field. The successful implementation of this application demonstrates a significant stride towards innovative educational tools in electrical engineering, fostering a deeper comprehension and practical skills among students and professionals alike.

Appendices:Python Script for Transmission Line Calculator

```python
from PySide6.QtWidgets import QApplication, QMainWindow, QLabel, QComboBox, QLineEdit,
    QPushButton, QMessageBox
from PySide6.QtCore import Qt
import math

epsilon_0 = 8.854187817620389e-12

# Define tower types and their specifications
tower_types = {
    "Type-1": {
        "max_height": 39,
        "min_height": 23,
        "max_horizontal_distance": 4,
        "min_horizontal_distance": 2.2,
        "voltage_level": 66,
        "max_conductors_bundle": 3
    },
    "Type-2": {
        "max_height": 43,
        "min_height": 38.25,
        "max_horizontal_distance": 11.5,
        "min_horizontal_distance": 9.4,
        "max_horizontal_distance_center_phase": 8.9,
        "voltage_level": 400,
        "max_conductors_bundle": 4
    },
    "Type-3": {
        "max_height": 48.8,
        "min_height": 36,
        "max_horizontal_distance": 5.35,
        "min_horizontal_distance": 1.8,
        "voltage_level": 154,
        "max_conductors_bundle": 3
    }
}

# Define conductor types and their specifications
conductor_types = {
    "Hawk": {"diameter": 21.793, "gmr": 8.809, "ac_resistance": 0.132, "current_capacity
        ": 659},
```

```
    "Drake": {"diameter": 28.143, "gmr": 11.369, "ac_resistance": 0.080, "
        current_capacity": 907},
    "Cardinal": {"diameter": 30.378, "gmr": 12.253, "ac_resistance": 0.067, "
        current_capacity": 996},
    "Rail": {"diameter": 29.591, "gmr": 11.765, "ac_resistance": 0.068, "
        current_capacity": 993},
    "Pheasant": {"diameter": 35.103, "gmr": 14.204, "ac_resistance": 0.051, "
        current_capacity": 1187}
}

def calculate_distance(coord1, coord2):
    x1, y1 = coord1
    x2, y2 = coord2
    distance = ((x2 - x1) ** 2 + (y2 - y1) ** 2) ** 0.5
    return distance

def calculate_gmr_bundle(num_conductors, gmr_conductor, distance):
    if num_conductors == 1:
        return gmr_conductor / 1000
    elif num_conductors == 2:
        return ((gmr_conductor / 1000) * distance) ** 0.5
    elif num_conductors == 3:
        return ((gmr_conductor / 1000) * distance * distance) ** (1 / 3)
    elif num_conductors == 4:
        return ((gmr_conductor / 1000) * distance * distance * distance * (2 ** 0.5)) **
             (1 / 4)
    else:
        return None

def calculate_req_bundle(number_conductor, d, distance):
    if number_conductor == 1:
        return d / 2000
    elif number_conductor == 2:
        return ((d / 2000) * distance) ** 0.5
    elif number_conductor == 3:
        return ((d / 2000) * distance * distance) ** (1 / 3)
    elif number_conductor == 4:
        return ((d / 2000) * distance * distance * distance * (2 ** 0.5)) ** (1 / 4)
    else:
        return None

class TransmissionLineCalculatorApp(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Transmission Line Calculator")
        self.setGeometry(100, 100, 700, 550)
        self.setup_ui()

    def setup_ui(self):
        self.instruction_label = QLabel("NOTE: Coordinates should be given as x,y
            without parenthesis. If single-circuit is used, enter prime coordinates as
            trivial 0,0.", self)
        self.instruction_label.setGeometry(20, 10, 800, 20)

        self.tower_label = QLabel("Select Tower Type:", self)
        self.tower_label.setGeometry(20, 40, 130, 30)
        self.tower_combobox = QComboBox(self)
        self.tower_combobox.addItems(list(tower_types.keys()))
        self.tower_combobox.setGeometry(150, 40, 200, 30)

        self.circuit_label = QLabel("Number of Circuits:", self)
        self.circuit_label.setGeometry(20, 90, 130, 30)
        self.circuit_entry = QLineEdit(self)
        self.circuit_entry.setGeometry(150, 90, 200, 30)

        self.phase_a_label = QLabel("Phase A Coordinates X,Y:", self)
        self.phase_a_label.setGeometry(20, 140, 180, 30)
        self.phase_a_entry = QLineEdit(self)
        self.phase_a_entry.setGeometry(220, 140, 130, 30)

        self.phase_aprime_label = QLabel("Phase A' Coordinates X,Y:\n(Enter 0,0 if
            single-circuit)", self)
        self.phase_aprime_label.setGeometry(380, 140, 180, 30)
```

```
        self.phase_aprime_entry = QLineEdit(self)
        self.phase_aprime_entry.setGeometry(560, 140, 130, 30)

        self.phase_b_label = QLabel("Phase B Coordinates X,Y:", self)
        self.phase_b_label.setGeometry(20, 190, 180, 30)
        self.phase_b_entry = QLineEdit(self)
        self.phase_b_entry.setGeometry(220, 190, 130, 30)

        self.phase_bprime_label = QLabel("Phase B' Coordinates X,Y:\n(Enter 0,0 if
            single-circuit)", self)
        self.phase_bprime_label.setGeometry(380, 190, 180, 30)
        self.phase_bprime_entry = QLineEdit(self)
        self.phase_bprime_entry.setGeometry(560, 190, 130, 30)

        self.phase_c_label = QLabel("Phase C Coordinates X,Y:", self)
        self.phase_c_label.setGeometry(20, 240, 180, 30)
        self.phase_c_entry = QLineEdit(self)
        self.phase_c_entry.setGeometry(220, 240, 130, 30)

        self.phase_cprime_label = QLabel("Phase C' Coordinates X,Y:\n(Enter 0,0 if
            single-circuit)", self)
        self.phase_cprime_label.setGeometry(380, 240, 180, 30)
        self.phase_cprime_entry = QLineEdit(self)
        self.phase_cprime_entry.setGeometry(560, 240, 130, 30)

        self.conductors_label = QLabel("Number of Conductors in the Bundle:", self)
        self.conductors_label.setGeometry(20, 290, 200, 30)
        self.conductors_entry = QLineEdit(self)
        self.conductors_entry.setGeometry(250, 290, 100, 30)

        self.distance_label = QLabel("Distance Between Conductors (in meters):", self)
        self.distance_label.setGeometry(20, 340, 250, 30)
        self.distance_entry = QLineEdit(self)
        self.distance_entry.setGeometry(270, 340, 80, 30)

        self.conductor_label = QLabel("Select Conductor Type:", self)
        self.conductor_label.setGeometry(20, 390, 150, 30)
        self.conductor_combobox = QComboBox(self)
        self.conductor_combobox.addItems(list(conductor_types.keys()))
        self.conductor_combobox.setGeometry(180, 390, 170, 30)

        self.length_label = QLabel("Length of Transmission Line (in km):", self)
        self.length_label.setGeometry(20, 440, 220, 30)
        self.length_entry = QLineEdit(self)
        self.length_entry.setGeometry(250, 440, 100, 30)

        self.calculate_button = QPushButton("Calculate", self)
        self.calculate_button.setGeometry(300, 490, 150, 40)
        self.calculate_button.clicked.connect(self.calculate_parameters)

    def calculate_parameters(self):
        tower_type = self.tower_combobox.currentText()
        circuit_number = int(self.circuit_entry.text())
        phase_a_coord = tuple(map(float, self.phase_a_entry.text().split(',')))
        phase_b_coord = tuple(map(float, self.phase_b_entry.text().split(',')))
        phase_c_coord = tuple(map(float, self.phase_c_entry.text().split(',')))
        phase_aprime_coord = tuple(map(float, self.phase_aprime_entry.text().split(','))
            )
        phase_bprime_coord = tuple(map(float, self.phase_bprime_entry.text().split(','))
            )
        phase_cprime_coord = tuple(map(float, self.phase_cprime_entry.text().split(','))
            )

        number_conductor = int(self.conductors_entry.text())
        distance = float(self.distance_entry.text())

        conductor_type = self.conductor_combobox.currentText()
        gmr_conductor = conductor_types[conductor_type]["gmr"]
        diameter_conductor = conductor_types[conductor_type]["diameter"]
        res_conductor = conductor_types[conductor_type]["ac_resistance"]

        gmr_bundle = calculate_gmr_bundle(number_conductor, gmr_conductor, distance)
        req_bundle = calculate_req_bundle(number_conductor, diameter_conductor, distance
```

```
            )

        phase_a_dist = calculate_distance(phase_a_coord, (0, 0))
        phase_b_dist = calculate_distance(phase_b_coord, (0, 0))
        phase_c_dist = calculate_distance(phase_c_coord, (0, 0))

        inductance = (2 * 10**-7) * math.log(phase_a_dist / gmr_bundle)
        capacitance = (2 * math.pi * epsilon_0) / math.log(req_bundle / phase_a_dist)

        QMessageBox.information(self, "Calculated Parameters",
                                f"GMR of the Bundle: {gmr_bundle:.6f} m\n"
                                f"Req of the Bundle: {req_bundle:.6f} m\n"
                                f"Inductance per meter: {inductance:.6f} H/m\n"
                                f"Capacitance per meter: {capacitance:.6f} F/m")

if __name__ == "__main__":
    app = QApplication([])
    window = TransmissionLineCalculatorApp()
    window.show()
    app.exec()
```